lrgpr: Low Rank Gaussian Process Regression

Gabriel E. Hoffman*

lrgpr version 0.0.6 as of April 18, 2014

Abstract

The **lrgpr** package provides an interactive R interface for fitting linear mixed models, also known as Gaussian Process Regression, on very large datasets. The package provides user-friendly interfaces for the linear mixed model and a computationally efficent extension termed Low Rank Gaussian Process Regression, as well as standard linear and logistic regression models. **lrgpr** allows fitting millions of regression models on a desktop computer by using an efficient implementation, parallelization and out-of-core computing for datasets that are too large to fit in main memory.

Contents

*Icahn Institute for Genomics and Multiscale Biology, Department of Genetics and Genomic Sciences, Icahn School of Medicine at Mount Sinai, New York, New York, USA

1 Overview

Genome-wide association studies (GWAS) are a widely used to identify genetic variants that are associated with variation in a phenotype of interest. Tests of association usually take the form of statistical hypothesis tests in a regression model that accounts for known confounding variables such as sex. Yet due to the complex nature of GWAS datasets, more sophisticated methods are used in order to maintain power while controling the false positive rate. The linear mixed model is the state-of-the-art method to account for the confounding effects of kinship and population structure in GWAS analysis, and much recent work as focused on this approach (Hoffman, 2013; Listgarten etal., 2012; Lippert etal., 2011; Kang etal., 2010; Zhou and Stephens, 2012).

To date, most software for GWAS data has been designed for a "one-size-fits-all" analysis. Yet as GWAS datasets have become increasing complex and heterogeneous, there is growing potential for interactive, exploratory data analysis. The **lrgpr** package privdes a user-friendly, interactive and computationally efficient analysis framework that faciliates custom, exploratory analysis of large GWAS datasets.

The **lrgpr** package provides:

- Seamless, interactive R interface to aribtarily large datasets through **bigmemory**'s `big.matrix`
- Scalable linear or logistic regression for millions of hypothesis tests using `glmApply`
- Fitting a full or low rank linear mixed model with `lrgpr`
- Data-adaptive construction of the genetic similarity matrix for the linear mixed model
- Scalable linear mixed model regression for millions of hypothesis tests using `lrgprApply`
- Ability to define arbitrary interaction models and perform composite hypothesis tests with `glmApply`, `lrgpr` and `lrgprApply`

2 Dependencies and installation

lrgpr has some dependencies that may not be installed on your system. Installation requires the GNU Scientific Library (GSL) and Boost C++ libraries. On Ubuntu these can be automatically installed:

```
shell> sudo apt-get install libboost-all-dev libgsl0-dev
```

On Redhat there should be a similar command using `yum`. Alternatively, you can install them manually from source:

- GSL: http://www.gnu.org/software/gsl/
- Boost: http://www.boost.org/users/download/

lrgpr also depends on a number of R packages that can be installed from CRAN:

```
> pkgs = c("Rcpp", "RcppGSL", "RcppProgress", "MASS", "doParallel",
"formula.tools", "BH", "bigmemory", "biganalytics", "aod")
> install.packages(pkgs)
```

lrgpr requires bigmemory >= v4.4.7, so install this from R-Forge:

```
> install.packages("bigmemory", repos="http://R-Forge.R-project.org")
```

Once the dependencies are installed, you can install **lrgpr**:

```
shell> R CMD INSTALL lrgpr_0.0.6.tar.gz
```

2.1 Accelerating linear algebra in R

Linear algebra operations are the bottleneck for **lrgpr** and many other packages in R. Compiling R with a good implementation of the BLAS linear algebra library can speed up some operations by >10x, and using a parallel library can further improve performance compared to the standard "reference BLAS" used by default. Using the best library for your machine, such as ATLAS (generic), ACML (AMD) or MKL (Intel), is strongly suggested.

You can follow the instructions to compile R against one of these libraries: http://cran.r-project.org/doc/manuals/R-admin.html#Linear-algebra

3 Using the `big.matrix` format

R typically stores all data in memory (i.e. RAM) for fast access. However, GWAS datasets can easily be too large to store in memory. For example, a dataset with 10,000 individuals and 1,000,000 SNPs would require $\sim$ 80 Gb to store in R in its standard format (i.e. as a `double`), and much larger datasets are now very common. Instead, **lrgpr** uses **bigmemory**'s `big.matrix` format to store a very large dataset in binary form directly on the hard drive and avoid using RAM while still allowing interactive access to the data. Thus the `big.matrix` format allows analysis of arbitrarily large datasets on any machine with sufficient hard drive capacity.

Arbitrary data can be converted to `big.matrix` format using **bigmemory**'s `write.big.matrix`. Alternatively, large GWAS data can be converted more efficently:

```
> path = system.file(package = 'lrgpr')
> tped_file = paste(path, "/extdata/test.tped", sep="")
> fam_file = paste(path, "/extdata/test.tfam", sep="")
> map_file = paste(path, "/extdata/test.map", sep="")
```

```
> convertToBinary(tped_file, "./test.binary", "TPED")
```

Supported GWAS data formats include TPED, GEN and DOSAGE.

Data is loaded into R by pointing a variable to the description file:

```
> X <- attach.big.matrix("./test.binary_descr")
```

Note that this data is not loaded into memory. Instead `X` points to the location of the data on the hard drive, and the data is loaded into memory only when the user asks for it. The user can treat `X` as a standard R `matrix` with the caveat that user must be carefull not to load too much data into memory at the same time. For example, the 10^{th} SNP can be accessed seamlessly:

```
> X[,10]
```

and the user can perform abitrary operations on each column individually without loading the whole dataset at the same time:

```
> column_means = c()
> for(j in 1:ncol(X)){
          column_means[j] = mean(X[,j])
 }
```

Accessing the data in this way creates a standard R `matrix` that can be passed to any function. Thus the user can process `X` as if it were stored in memory; the complexity of storing the data on the hard drive is completely hidden from the user.

Processing the entire dataset in R can be costly for large datasets, so **lrgpr** provides functions to report common statistics more efficiently by performing all computations in C/C++:

```
> freq = getAlleleFreq(X)
```

4 Regression with R's `glm`

R fits generalized linear models using the `glm` function that provides flexability in defining the regression model. I describe `glm` very briefly here so that I can later describe how **lrgpr** mirrors this functionality. For example, we can read Plink's FAM file and fit a simple model:

```
> FAM = read.fam(fam_file)
> y = FAM$phenotype
> sex = FAM$sex
```

```
> fit = glm( y ~ sex )
```

We can see the model coefficients and statistics:

```
> fit
```

```
> summary(fit)
```

and plot model diagnostics:

```
> par(mfrow=c(2,2))
> plot(fit)
```

The user can also specify and perform hypothesis tests on more complex models involving interations:

```
> fit = glm( y ~ sex + X[,10]*X[,100])
```

```
> wald.test(vcov(fit), coef(fit), Terms=3:5)
```

5 Fitting many regressions with `glmApply`

Analysis of GWAS data involves fitting a regression model for each SNP individually. The user could write a simple `for` loop that calls the `glm` function, but the computation can be accelerated by doing the calculation in the C/C++ backend and reporting the results.

A standard analysis can be performed with **lrgpr**'s `glmApply`:

```
> pValues = glmApply( y ~ SNP + sex, features=X, terms=2)$pValues
```

where the `formula` follows standard `glm` syntax, `features` specifies the `big.matrix` or standard R `matrix` with columns as genetic markers, and `terms` specifies which variables from the `formula` are in the hypothesis test. Note that `SNP` is a placeholder for each sucessive column in `features`, and if `terms` is omitted the hypothesis test includes all terms corresponding to `SNP`:

```
> pValues = glmApply( y ~ SNP + sex, features=X)$pValues
```

This code is the same as:

```
> pValuesR = c()
> for(j in 1:ncol(X)){
          # fit model
          fit = glm(y ~ X[,j] + sex)

          # perform hypothesis test
          pValuesR[j] = wald.test(vcov(fit), coef(fit),
Terms=2)$result$chi2[3]
 }
```

but calling `glmApply` is simpler and much faster for large datasets.

Note that `glmApply` can also efficiently fit multivariate regression models with multiple response variables. See `?glmApply` for details.

6 Linear mixed models with `lrgpr`

The `lrgpr` function fits full and low rank linear mixed models and follows the syntax and behavior of `glm`.

The linear mixed model has the form:

$$\mathbf{y} \sim \mathcal{N}(\mathbf{X}\boldsymbol{\beta}, \mathbf{K}\sigma_a^2 + \mathbf{I}\sigma_e^2),$$

where $\mathbf{y}$ $(n \times 1)$ is the vector of phenotype values, n is the sample size, $\mathbf{X}$ $(n \times c)$ is the design matrix of c fixed effects, $\boldsymbol{\beta}$ $(c \times 1)$ is the vector of coefficients, $\mathbf{K}$ $(n \times n)$ is the genetic similarity matrix (GSM), $\mathbf{I}$ $(n \times n)$ is the identity matrix, σ_a^2 is the magnitude of the genetic variance, and σ_e^2 is the magnitude of the residual variance. The parameters are estimated by maximizing the log-loglihood using the algorithms of Lippert etal. (2011) and Listgarten etal. (2012).

When $\mathbf{K}$ is full rank, the coefficient estimates and their sample variance are, respectively,

$$\begin{aligned} \hat{\boldsymbol{\beta}} &= (\mathbf{X}^T\boldsymbol{\Omega}\mathbf{X})^{-1}\mathbf{X}^T\boldsymbol{\Omega}\mathbf{y} \\ \hat{\boldsymbol{\Sigma}} &= (\mathbf{X}^T\boldsymbol{\Omega}\mathbf{X})^{-1}\sigma_a^2 \end{aligned}$$

where $\boldsymbol{\Omega} = (\mathbf{K} + \mathbf{I}\frac{\sigma_e^2}{\sigma_a^2})^{-1}$ and the values of σ_a^2 and σ_e^2 are already estimated by maximum likelihood. Following standard likelihood theory, the Wald test is

$$\hat{\boldsymbol{\beta}}_h^T(\hat{\boldsymbol{\Sigma}}_h)^{-1}\hat{\boldsymbol{\beta}}_h \sim \chi^2_{|h|}$$

where h specifies the fixed effects being tested and $|h|$ is the number of entries. When $\mathbf{K}$ is not full rank the estimates and hypothesis tests are analogous and follow directly from Listgarten etal. (2012).

6.1 The genetic similarity matrix and its spectral decomposition

Following the algorithms of Lippert etal. (2011) and Listgarten etal. (2012), the genetic similarity matrix enters the linear mixed model only through its spectral decomposition. There are multiple ways to compute $\mathbf{K}$ or its spectral decomposition.

6.1.1 Full rank linear mixed model

When the genetic similarity matrix, $\mathbf{K}$, is based on a genome-wide set of markers, the linear mixed model is full rank. A number of similarity metrics are widely used (Yang etal., 2011; Kang etal., 2010;

Zhou and Stephens, 2012) and a genetic similarity matrix can easily be imported into R from another program:

```
> K = read.table( paste(path, "/extdata/K.txt", sep="") )
> dcmp = eigen(K, symmetric=TRUE)
```

However, only the spectral decomposition of $\mathbf{K}$ is needed, and this can be computed directly in R:

```
> index = seq(1, ncol(X), by=100)
```

```
> v = getAlleleVariance(X[,index])
> j = index[which(v > .01)]
```

```
> dcmp = svd(scale(set_missing_to_mean(X[,j])))
```

We can then fit the linear mixed model using the same syntax as for `glm`:

```
> fit = lrgpr(y ~ sex, dcmp)
```

Features of lrgpr

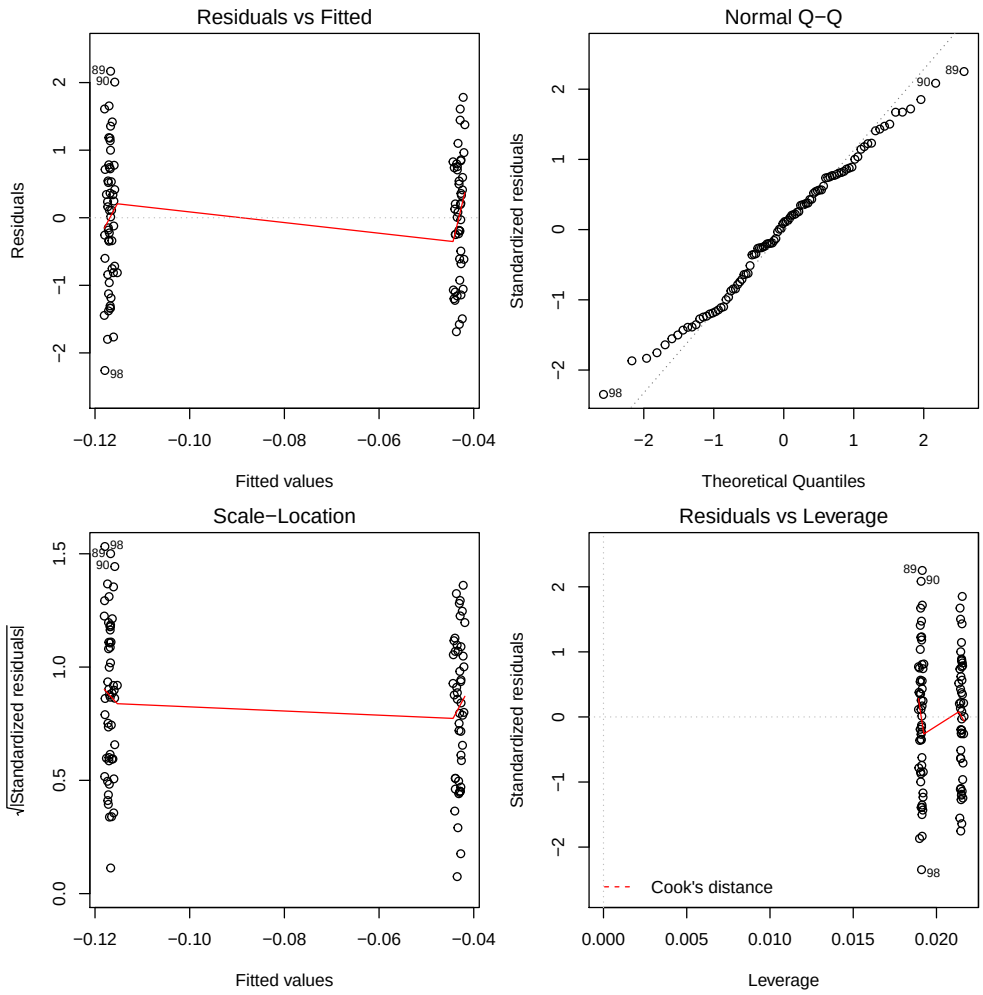
Since `lrgpr` behaves like `glm` and we can see the model coefficients and statistics:

```
> fit
Call:
lrgpr(formula = y ~ sex, decomp = dcmp)

Coefficients:
(Intercept)         sex2
-0.11672805   0.07347214

Variance components:
     sigSq_a      sigSq_e         delta
4.205881e-05 9.264070e-01 2.202647e+04
```

```
> summary(fit)
Call:
lrgpr(formula = y ~ sex, decomp = dcmp)

Residuals:
     Min       1Q   Median       3Q      Max
-2.26046 -0.76852  0.09567  0.72788  2.16791

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.11673     0.1322 -0.8828   0.3773
sex2         0.07347     0.1929  0.3809   0.7033
--
Signif. codes:  0 "***" 0.001 "**" 0.01 "*" 0.05 "." 0.1 " " 1

Residual standard error: 0.9723 on 97.98 degrees of freedom
Multiple R-squared: 0.001673 , Adjusted R-squared: -0.01893


Variance components:
     sigSq_a      sigSq_e        delta
4.205881e-05 9.264070e-01 2.202647e+04
```

and plot model diagnostics:

```
> fitDiag = lrgpr(y ~ sex, dcmp, diagnostic=TRUE)
> par(mfrow=c(2,2), mar=c(4,4.3,2,1))
> plot(fitDiag)
```



The user can also specify and perform hypothesis tests on more complex models involving interations:

```
> fit = lrgpr( y ~ sex + X[,10]*X[,100], dcmp)
```

```
> wald(fit, terms=3:5)
```

This functionality can be used to perform custom, exporatory analysis on arbitrarily large datasets.

6.1.2 Low rank linear mixed model: (low rank gaussian process regression)

The full rank linear mixed model can be very computationally expensive for large datasets and a low rank version was proposed to address this issue (Lippert etal., 2011; Listgarten etal., 2012). A low rank GSM, or more precisely its spectral decomposition, can be constructed by using fewer SNPs than the sample size. Instead of capturing the genome-wide similarity, the GSM can be constructed based on the k SNP's that are most correlated with the phenotype. Following Lippert etal. (2011) and Listgarten etal. (2012), we construct the GSM using the most signficant markers from an uncorrected association test:

```
> pValues = glmApply( y ~ SNP + sex, features=X, terms=2)$pValues
```

```
> ord = order(pValues, decreasing=FALSE)
```

Now construct the spectral decomposition from the top $k = 100$ SNPs and fit the low rank linear mixed model:

```
> k = 100
> dcmp = svd(scale(set_missing_to_mean(X[,ord[1:k]])))
```

```
> fit = lrgpr(y ~ sex, dcmp)
```

Learning the optimal rank of the low rank linear mixed model

We arbitrarily selected the top $k = 100$ SNPs above, but following Listgarten etal. (2012) we can use cross-validation to select k based on the data. Here the cross-validation error is calculated for multiple values of k:

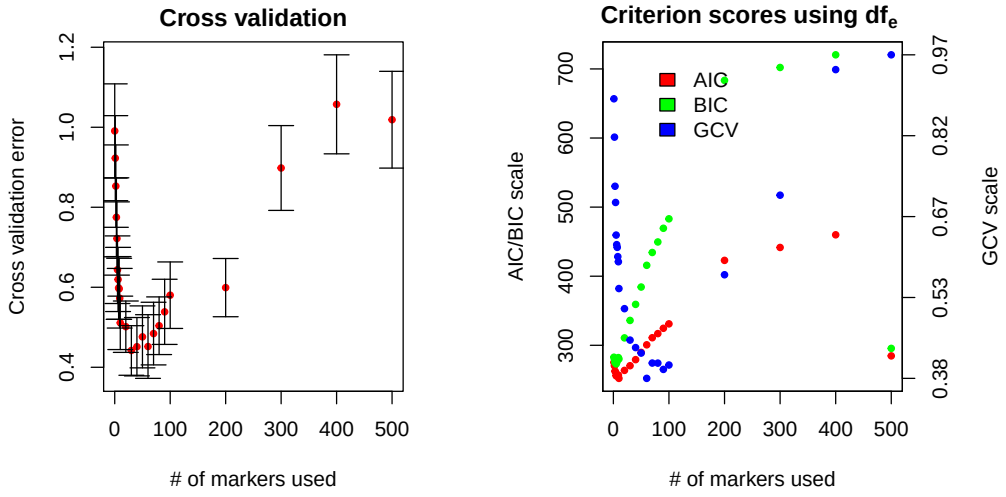
```
> fitcv = cv.lrgpr( y ~ sex, features=X, order=ord, nfolds=2)
```

Instead of performing computationally expensive cross-validation, we can use model criterion such as Akaike Information Criterion (AIC), Bayesian Information Criterion (BIC) or Generalized Cross-Validation (GCV) to select k based on the degrees of freedom for the linear mixed model described by Hoffman (2013):

```
> fitcrit = criterion.lrgpr( y ~ sex, features=X, order=ord )
```

We can plot the cross-validation error and model criterion scores:

```
> par(mfrow=c(1,2))
> plot.cv.lrgpr(fitcv)
> plot.criterion.lrgpr(fitcrit)
```



and compute to spectral decomposition of the GSM based on the set of SNPs selected by Generalized Cross-validation:

```
> k = fitcrit$best$GCV
> dcmp = svd(scale(set_missing_to_mean(X[,ord[1:k]])))
```

Note that the GCV score is very good proxy for the cross-validation error and most of the difference between these values are driven primarily by the finite sampling in cross-validation.

6.2 Proximal contamination

If the random effect includes the marker to be tested, the correlation between the two can decrease power. More generally, the inclusion of a marker from the same linkage-disequilibrium (LD) block as the marker being tested can decrease power. This is termed *proximal contamination* and Listgarten etal. (2012) proposed an efficient solution. Intuitively, we would like to use a random effect that excludes any markers from the same LD block as the one being tested. This can be used by dropping a set of markers from the GSM and recomputing the spectral decomposition. However, this can be very computationally expensive and we use the algorithm of Listgarten etal. (2012) which reuses the same spectral decompution but which omits the set of proximal contamination markers by modifing the log-loglikelihood calculations of `lrgpr`.

Here, `lrgpr` is evaluated by using markers `2:5` in the random effect and testing marker 1.

```
> dcmp = svd(scale(set_missing_to_mean(X[,2:5])))
> fit = lrgpr(y ~ X[,1], decomp=dcmp)
> summary(fit)$coefficients
               Estimate Std. Error    t value  Pr(>|t|)
(Intercept) -0.03198182  0.1615661 -0.1979488 0.8430851
X[, 1]      -0.04523813  0.1169069 -0.3869587 0.6987868
```

If the spectral decomposition includes marker 1, then proximal contamination is an issue. We can pass marker 1 into the `W_til` argument to address this.

```
> dcmp = svd(scale(set_missing_to_mean(X[,1:5])))
> fit = lrgpr(y ~ X[,1], decomp=dcmp, W_til=scale(X[,1]))
> summary(fit)$coefficients
               Estimate Std. Error    t value  Pr(>|t|)
(Intercept) -0.03198182  0.1615661 -0.1979488 0.8430851
X[, 1]      -0.04523813  0.1169069 -0.3869587 0.6987868
```

The results of the two evaluations are exactly the same, with the second having the advantage of avoiding the issue of decreased power due to proximal contamination without having to recompute the spectral decomposition and exclude each marker in turn.

7 Fitting many `lrgpr` regressions with `lrgprApply`

The functionality of `lrgprApply` is the focus of the **lrgpr** package. Once the spectral decomposition has been computed, we can perform a genome-wide regression analysis with `lrgprApply` which is analogous to `glmApply`:

```
> pValuesLMM = lrgprApply( y ~ SNP + sex, features=X, decomp=dcmp,
terms=2)
```

Moreover, `lrgprApply` can be used to fit a regression model with interactions and the `terms` can be set to perform the appropriate hypothesis test:

```
> pv1 = lrgprApply( y ~ SNP*sex, features=X, decomp=dcmp, terms=4)
```

```
> pv2 = lrgprApply( y ~ sex + SNP*X[,1], features=X, decomp=dcmp,
terms=3:5)
```

The user has a lot of flexabilty to design a custom analysis to address the specific question of his or her study.

7.1 Proximal contamination

We can also drop proximal markers in `lrgprApply` using either genetic or physical distance to define the genomic window. Using the map file from plink:

```
> MAP = read.table( map_file )
```

we can call perform a genome-wide analysis:

```
> pValuesLMMprox = lrgprApply( y ~ sex + SNP, features=X, decomp=dcmp,
terms=3, map=MAP[,c(1,3)], distance=2, dcmp_features=ord[1:k])
```

where entries in `MAP` correspond to the markers in `X`, `map=MAP[,c(1,3)]` indicates the marker names and their position on the genetic map, `distance` determines the size of the window corresponding to the locations in `MAP[,3]`, and `dcmp_features` indicates the indices of `X` used to construct `dcmp`. Following the standard format for plink map files, the physical location can be used by specifing `map=MAP[,c(1,4)]` and setting `distance` appropriately.

7.1.1 Interaction model

When evaluating an interaction model with the formula `y ~ sex + SNP*X[,1]`, the syntax above addresses the proximal contamination due to `SNP`, but not due to `X[,1]`. We can include additional markers in the proximal contamination set by also specifying `W_til`:

```
> pv2 = lrgprApply( y ~ sex + SNP*X[,1], features=X, decomp=dcmp,
terms=3:5, map=MAP[,c(1,3)], distance=2, dcmp_features=ord[1:k],
W_til=X[,1])
```

8 Implementation

The **lrgpr** package provides an R interface to high performance computational statistics code written in C/C++ that builds off the GNU Scientific Library (GSL). Linear algebra operations are performed by BLAS and LAPACK and the performance is highly dependent on the library installed on the local machine. Data is passed from R to the C++ backend using **Rcpp** and **RcppGSL**. Parallelization is performed by OpenMP. Out-of-core processing is faciliated by **bigmemory**, which stores the full dataset on the hard drive instead of in memory and uses `mmap` from the GNU C Library for random data access.

References

Hoffman, G.E., 2013. Correcting for Population Structure and Kinship Using the Linear Mixed Model: Theory and Extensions. PLoS ONE 8(10), e75707.

Kang, H.M., Sul, J.H., Service, S.K., Zaitlen, N.A., Kong, S.-y. etal., 2010. Variance component model to account for sample structure in genome-wide association studies. Nature Genetics 42(4), 348–54.

Lippert, C., Listgarten, J., Liu, Y., Kadie, C.M., Davidson, R.I. etal., 2011. FaST linear mixed models for genome-wide association studies. Nature Methods 8(10), 833–5.

Listgarten, J., Lippert, C., Kadie, C.M., Davidson, R.I., Eskin, E. etal., 2012. Improved linear mixed models for genome-wide association studies. Nature Methods 9(6), 525–6.

Yang, J., Lee, S.H., Goddard, M.E., and Visscher, P.M., 2011. GCTA: a tool for genome-wide complex trait analysis. American Journal of Human Genetics 88(1), 76–82.

Zhou, X. and Stephens, M., 2012. Genome-wide efficient mixed-model analysis for association studies. Nature Genetics 44(7), 821–4.